RESEARCH ARTICLE OPEN ACCESS

Feature Design of Maintainable Web application Using React js

ARPIT KUMAR

Computer Science and Engineering, RV College of Engineering, and Bangalore

Email: arpitkumar.cs17@rvce.edu.in)

--************************----------------------------------

Abstract:

Because of the increasing demand for advanced functionality as well as the short time-to-market, web application maintainability is becoming increasingly important. Fixing errors, reusing functionality, and efficiently adding new features are critical for making the application profitable for the software company as well as valuable for the end user. Modern frameworks and libraries, such as React, help web developers create sophisticated applications by utilizing high-quality solutions known as architectural patterns. The primary goal of this paper is to use the standard design system framework for UI redesign and the development of the target digital product. The purpose of this redesign is to increase UI consistency and quality, making the design and development process more efficient while also allowing designers and developers to share a common vocabulary.

Keywords **—MVC, GUI, React js, Virtual DOM.**

--************************----------------------------------

I. INTRODUCTION

Architectural patterns have evolved as web applications have progressed from simple web pages to sophisticated corporate products, providing web engineers with high-quality solutions to recurring problems. Web developers must stay up to date on new technologies and try them out. Given that ReactJS is one of them, it's no wonder that it's sweeping the codebase. ReactJS is a frontend development library written in JavaScript that is open-source. Users nowadays want faster and more dynamic webpages, while developers need a modern and flexible development environment free of boilerplate. As a result, ReactJS [1] is rising in popularity among frontend developers. ReactJS is a frontend development library written in JavaScript that is open-source. Users nowadays want faster and more dynamic webpages, while developers need a modern and flexible development environment free of boilerplate. As a result, ReactJS is rising in popularity among frontend developers [2]. The architectural pattern used for the client-side application may change depending on the design of these APIs [5].The setting is fast-paced, with agile development and quick iterations [3], as it is in many software development businesses. The software's maintainability is critical for establishing lucrative projects that satisfy both the customer and the end-user. As a result, maintaining the web application's dynamic and adaptability to new changes and features becomes a top priority.

A. React Hooks

React Hooks are used to connect more complex function components to React features. Hooks in React have a name convention that starts with the word "use." A function component does not have state by default; instead, a React hook named useState can be used to keep the state throughout the life of the component. All hooks, including useState, can be used multiple times within the same component.

B. Context API

The Context API is a built-in method for sharing data between React components without utilizing props-drilling. The createContext method in the

React library is used to create a Context that can be shared by numerous components in the component tree. A Provider on the Context instance is used to add data to the Context and make it available to children components. The Provider is a component with a single property named value that specifies the data for the Context, which can be variables, functions, or objects. Context data can be retrieved from any child component using a Consumer accessible on the Context instance by wrapping children components inside the Provider component. Instead of relying on a single Context instance, the application makes use of many contexts for various reasons.

C. *Model-View-Controller*

The MVC pattern is one of the earliest architectural patterns [8], and it was originally used to create GUIs for desktop programmes. It is made up of three parts: the Model (M), the View (V), and the Controller (C) (C). The MVC architecture's ultimate purpose is to decouple the user interface from the data that it represents.

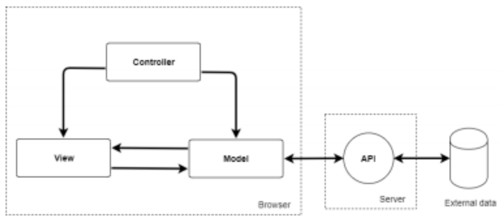

Figure. 1. Client-side MVC architectural pattern

D. *Redux In React*

React applications typically employ Redux as a data management solution, and a library called react-redux can be used to make the Redux store accessible from every React component. By linking the Redux store to the components, every component in the application can update and read data from the Redux store without having to use props drilling. The connect function in the react-redux library is used to extend React components with functionality for dispatching actions and reading state from the Redux store. The connect function must be able to access the application's instantiated Redux store, which is accomplished using the Provider component from the react-redux package.

II. PERFORMANCE

ReactJS is well-known for its outstanding performance. This is one of the fundamental characteristics that distinguishes the frameworks from the many other frameworks available in the serious sector. The virtual DOM feature of the framework is primarily responsible for the framework's highly effective execution. ReactJS does this by maintaining a virtual document [4] object model in memory. Rather of refreshing the browser DOM immediately after making a modification to the currently displayed website page, first modifications to virtual DOM are made. After making changes to the virtual DOM, a diff() algorithm is used, which considers the tow, the virtual DOM, and the browsing DOM, and only significant and wanted nodes of the programme DOM tree are updated, resulting in blazing fast application execution.

III. WORKING

The M-V-C plan stands for model view controller. In web applications as well as front-end apps running on any platform, worldview is common and important for UI improvement. If web-applications are present, DOM communicates with physical View. The DOM is created using an HTML layout that is derived from a different document, a content square, or a precompiled format work. The View element gives life to the printed layout as a DOM. As part of the document object model tree's life cycle, it assumes a major role in dealing with Events and controlling the document object model tree. A point of view is valuable if and only if it allows for client cooperation while also displaying the relevant information. Information is a type of data that is retrieved from a Data-Store, which can be a database, a web application, or a local store. Frameworks provide a way to connect the view to the data store, ensuring that database changes are automatically reflected in the view. Data Binding [12] is the term used to describe the process of pushing scheduled information updates. There are a slew of application programme interfaces, or APIs, that make this process a breeze. The M-V-C worldview is completed by the C component, such as the Controller, which draws in the other two components, such as the model and the view, and allows the information model stream into the View and client events out of View, causing changes in the Model.

To explore how React handles these tasks, you'll need to learn a lot more about components, starting with the Component. In React, the Component is the most important structure. The entire user interface can be planned by assembling a tree of several components. The render() technique creates an intermediate DOM that displays all of the React components. A Call to React, as depicted in Figure 2. The renderComponent() technique on the root Component causes the intermediate DOM to be produced by recursively travelling down the Component tree. After a while, this intermediary DOM appeared. React creates the component tree as a mix of multiple XML nodes using an advantageous XML-based expansion to JavaScript known as the JSX. This makes DOM representation and viewing much easier and more useful. In addition, JSX plays an important role in optimizing the interaction between event handlers and properties as XML characteristics. The final JavaScript is created with the help of an order line and an in-program device.

A JSX XML node maps a Component in a straightforward manner. It's important to remember that React functions independently of JSX, and that JSX's role is limited to decoupling the task of producing intermediate DOM. Life Cycle of a Component Every component in the ReactJS framework has a very precise lifetime and contains a state-machine with three distinct states. A Component's life is extended by the Mounting method. The component tree, also known as the Intermediate DOM, is created when mounting goes through a render-pass. After that, the tree is converted into a container node in the true document object model. When the React.RenderComponent() approach is called, the full procedure takes place.

When the state of a component is updated using the setState() strategy or the properties are modified using the setProps() approach, the component is refreshed. A call to the render()method follows, which synchronizes the archive object model with information like as props and state. Between refreshes, Respond calculates the difference between the previous component-tree and the freshly produced tree. This progression has been greatly improved, and a leader has been added to limit the true DOM control. Un-mounted is the final state. If a component that will in general be a child is no longer formed in a render() call, this will happen. Engineers frequently don't have to worry about this and simply let React do its job. It would have been a serious infraction if react had not educated when it switched between the Mounted-Update-Unmounted stages. In any event, such is not the case, and snares are provided that can be superseded to alert at any time a state change occurs.

IV. LIMITATIONS

React has a few limitations that should be examined before using it for any project development. These are the following: The React only deals with the View material in a versatile view controller or MVC. As a result, additional tooling is necessary to complete the project development. For a couple of developers, using inline formats and JSX can be a very unpredictable and exhausting task. Similarly, if ReactJS is used, disappointments occur at build time rather at runtime, as is the case with other languages and frameworks, which can be perplexing and tiresome at times.

V. CONCLUSIONS

Despite a few minor flaws, ReactJS is undeniably superior. The modern web is becoming more distinct and client-centric by the day. Patterns of client experience configuration are constantly evolving and changing. Customer contents now ensure that only the most important and fundamental information is distributed, and that a consistent and pleasing experience is maintained across the entire internet. Straightforwardness, effectiveness, and more conspicuous openness are in demand in today's environment. ReactJS has a lot of power and shines when it comes to meeting the requirements of current patterns. In general, it can be said that ReactJS will undoubtedly have an impact on the way online apps are built.

REFERENCES

[1] Sanchit Aggarwal, 'Modern Web-Development using ReactJS', International Journal of Recent Research Aspects ISSN: 2349-7688, Vol. 5, Issue 1, March 2018.

[2] Anurag Kumar and Ravi Kumar Singh, "COMPARATIVE ANALYSIS OF ANGULARJS AND REACTJS", "International Journal of Latest Trends in Engineering and Technology", vol. 25, no. 6, pp. 34–40, 2011.

[3] Arshad Javeed, "Performance Optimization Techniques for ReactJS", "International Conference on Electrical, Computer and Communication Technologies", vol. 37, no. 10, pp. 46–54, 2004.

[4] Khuat, Tung, "Developing a frontend application using ReactJS and Redux", "International Journal of Latest Trends in Engineering and Technology", vol. 28, no. 6, pp. 78–85, 2011.

[5] Annie Ying, Yunhui Zheng, Jim A. Laredo, "Opportunities in Software Engineering Research for Web API Consumption", "IEEE/ACM 1st International Workshop on API Usage and Evolution (WAPI)", vol. 85, no. 12, pp. 2840–2859, 2012.

[6] K. Bak, Z. Diskin, M. Antkiewicz, K. Czarnecki, and A. Wasowski, "Clafer: unifying class and feature modelling," Software and System Modelling, vol. 15, no. 3, pp. 811–845, 2016.

[7] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. "Software Engineering Metrics and Models". Redwood City, CA, USA: Benjamin-Cummings Publishing Co., Inc., 1986. ISBN: 0-8053- 2162-4.

[8] A. Leff and J. T. Rayfield. "Web-application development using the Model/View/- Controller design pattern". In: Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference. Sept. 2001, pp. 118–127. DOI: 10. 1109 / EDOC. 2001. 950428.

[9] C. Krupitzer, M. Breitbach, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker, "A survey on engineering approaches for self-adaptive systems (extended version)," 2018.

[10] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Design Patterns: Elements of Reusable Object-oriented Software. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995. ISBN: 0-201-63361-2.

[11] J. Floch, S. O. Hallsteinsen, E. Stav, F. Eliassen, K. Lund, and E. Gjørven, "Using architecture models for runtime adaptability," IEEE Software, vol. 23, no. 2, pp. 62–70, 2006.

[12] J. Conallen, "Modelling Web application architectures with UML," Communications of the ACM, vol. 42, pp. 63-70, 1999.

[13] D. Rodriguez, R. Harrison, and M. Satpathy, "A generic model and tool support for assessing and improving Web processes," in Proc. IEEE Symposium, 2002, pp. 141-151.

[14] M. E. Fayad, M. Laitinen, and R. P. Ward, "Thinking objectively: software engineering in the small," Communications of the ACM, vol. 43, pp. 115-118, 2000.